Building a C++ Reflection System

Using LLVM and Clang

Storytime

Wouldn't it be great …

```
struct User {
    uint64_t id;
    string name;
    vector<string> pets;
};

User user;
user.id = 42;
user.name = "John";
user.pets.push_back("Buddy");
user.pets.push_back("Cooper");

string json = json::Stringify(&user);
```

```
{
    "id": 42,
    "name": "John",
    "pets": ["Buddy", "Cooper"]
}
```

How do we do this?

```
string
json::Stringify(User const *user)
{
    JsonSerializer serializer;
    serializer.SerializeInt64("id", user->id);
    serializer.SerializeString("name", user->name);
    serializer.BeginArray("pets");
    for (auto const &pet : user->pets)
        serializer.ArrayAddString(pet);
    serializer.EndArray();
    return serializer.ToString();
}
```

```
class User
{
    public ulong id;
    public string name;
    public List<string> pets = new List<string>();
}
```

```
User user = new User();
user.id = 42;
user.name = "John";
user.pets.Add("Buddy");
user.pets.Add("Cooper");
string json = JsonConvert.SerializeObject(user);
```

Reflection

```
Type t = user.GetType();
```

- GetField(String, BindingFlags)
- GetFields(BindingFlags)
- GetInterface(String, Boolean)
- GetInterfaces()
- GetMethodImpl(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- GetMethods(BindingFlags)
- GetNestedType(String, BindingFlags)
- GetNestedTypes(BindingFlags)
- GetProperties(BindingFlags)
- GetPropertyImpl(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])
- HasElementTypeImpl()
- InvokeMember(String, BindingFlags, Binder, Object, Object[], ParameterModifier[], CultureInfo, String[])
- IsArrayImpl()
- IsByRefImpl()
- IsCOMObjectImpl()
- IsPointerImpl()
- IsPrimitiveImpl()

- GetField(String, BindingFlags)
- **GetFields(BindingFlags)**
- GetInterface(String, Boolean)
- GetInterfaces()
- GetMethodImpl(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- **GetMethods(BindingFlags)**
- GetNestedType(String, BindingFlags)
- GetNestedTypes(BindingFlags)
- GetProperties(BindingFlags)
- GetPropertyImpl(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])
- HasElementTypeImpl()
- InvokeMember(String, BindingFlags, Binder, Object, Object[], ParameterModifier[], CultureInfo, String[])
- IsArrayImpl()
- IsByRefImpl()
- IsCOMObjectImpl()
- IsPointerImpl()
- IsPrimitiveImpl()

```
Type t = user.GetType();
FieldInfo[] fields = t.GetFields(...);

foreach (var field in fields) {
    Console.WriteLine("Name: {0}", field.Name);
    Console.WriteLine("Type: {0}", field.FieldType);
    Console.WriteLine();
}
```

```
Name: id
Type: System.UInt64

Name: name
Type: System.String

Name: pets
Type: System.Collections.Generic.List`1[System.String]
```

Back to C+

```
Class const *c = GetClass<User>();

for (auto &field : c->Fields()) {
    printf("Name: %s\n", field.Name());
    printf("Type: %s\n", field.Type().Name());
    printf("\n");
}
```

Blueprint

```
struct Type {
    char const *name;
    size_t size;
};
```

```
struct Class : public Type {
    Field fields[N];
    Function functions[N];
};
```

```
struct Field {
    Type *type;
    char const *name;
    size_t offset;
};
```

```
struct Function {
    Field returnValue;
    Field parameters[N];
    char const *name;
};
```

```
struct Type {
    char const *name;
    size_t size;
};

struct Field {
    Type *type;
    char const *name;
    size_t offset;
};

struct Function {
    Field returnValue;
    Field parameters[N];
    char const *name;
};

struct Class : public Type {
    Field fields[N];
    Function functions[N];
};
```

Data?

```
struct User {
    uint64_t id;
    string name;
    vector<string> pets;
};
```

```
Class const *
GetClass<User>()
{
    static Class clazz;
    clazz.fields[0].type = GetType<uint64_t>();
    clazz.fields[0].name = "id";
    clazz.fields[0].offset = offsetof(User, id);
    clazz.fields[1].type = GetType<string>();
    clazz.fields[1].name = "name";
    clazz.fields[1].offset = offsetof(User, name);
    clazz.fields[2].type = GetType<vector<user>>();
    clazz.fields[2].name = "pets";
    clazz.fields[2].offset = offsetof(User, pets);
    return &clazz;
}
```

```
Undefined symbols for architecture x86_64:
  "Type const* GetType<std::__1::basic_string<char, std::__1::char_traits<char>,
    std::__1::allocator<char> > >()", referenced from:
      _main in Untitled 7-0fb8bc.o
  "Type const* GetType<std::__1::vector<std::__1::basic_string<char,
    std::__1::char_traits<char>, std::__1::allocator<char> >,
    std::__1::allocator<std::__1::basic_string<char, std::__1::char_traits<char>,
    std::__1::allocator<char> > > >()", referenced from:
      _main in Untitled 7-0fb8bc.o
  "Type const* GetType<unsigned long long>()", referenced from:
      _main in Untitled 7-0fb8bc.o
ld: symbol(s) not found for architecture x86_64
```

"Primitive" Types

```
template<>
Type const *
GetType<int>()
{
    static Type t{"int", sizeof(int)};
    return &t;
}
```

```
template<class T>
Type const *
GetType()
{
    return detail::GetTypeImpl(TypeTag<T>{});
}
```

```
template<class T>
Type const *
GetTypeImpl(TypeTag<vector<T>>)
{
    /* ... */
}
```

```
Class const *
GetClassImpl(ClassTag<User>)
{
    static Class clazz;
    clazz.fields[0].type = GetType<uint64_t>();
    clazz.fields[0].name = "id";
    clazz.fields[0].offset = offsetof(User, id);
    clazz.fields[1].type = GetType<string>();
    clazz.fields[1].name = "name";
    clazz.fields[1].offset = offsetof(User, name);
    clazz.fields[2].type = GetType<vector<user>>();
    clazz.fields[2].name = "pets";
    clazz.fields[2].offset = offsetof(User, pets);
    return &clazz;
}
```

```
Class const *c = GetClass<User>();

for (auto &field : c->Fields()) {
    printf("Name: %s\n", field.Name());
    printf("Type: %s\n", field.Type().Name());
    printf("\n");
}
```

```
Name: id
Type: uint64_t

Name: name
Type: std::string

Name: pets
Type: std::vector<std::string>
```

LLVM

Clang

LibTooling

How?

Hello, AST

```
struct Foo {
    volatile int bar;
    float baz;
};
```

```
% clang -Xclang -ast-dump -fsyntax-only foo.h

TranslationUnitDecl 0x7ff8b00264d0 <<invalid sloc>> <invalid sloc>
`-RecordDecl 0x7f9f2a827120 <foo.h:1:1, line:4:1> line:1:8 struct Foo definition
  |-FieldDecl 0x7f9f2a877400 <line:2:5, col:18> col:18 bar 'volatile int'
  `-FieldDecl 0x7f9f2a877460 <line:3:5, col:11> col:11 baz 'float'
```

libTooling AST Visitor

```
struct DumpASTAction : public ASTFrontendAction
{
    std::unique_ptr<ASTConsumer>
    CreateASTConsumer(CompilerInstance &ci, StringRef inFile) override
    {
        return clang::CreateASTDumper(
            nullptr,/* dump to stdout */
            "",     /* no filter */
            true,   /* dump decls */
            true,   /* deserialize */
            false   /* don't dump lookups */
            );
    }
};

static llvm::cl::OptionCategory gToolCategory("metareflect options");
int main(int argc, char **argv)
{
    CommonOptionsParser optionsParser(argc, argv, gToolCategory);
    ClangTool tool(optionsParser.getCompilations(), optionsParser.getSourcePathList());
    return tool.run(newFrontendActionFactory<DumpASTAction>().get());
}
```

```
% ./metareflect foo.h -- -I. $CFLAGS

TranslationUnitDecl 0x7ff8b00264d0 <<invalid sloc>> <invalid sloc>
`-RecordDecl 0x7f9f2a827120 <foo.h:1:1, line:4:1> line:1:8 struct Foo definition
  |-FieldDecl 0x7f9f2a877400 <line:2:5, col:18> col:18 bar 'volatile int'
  `-FieldDecl 0x7f9f2a877460 <line:3:5, col:11> col:11 baz 'float'
```

```
#include <stdint.h>
#include <vector>
#include <string>

struct User
{
    uint64_t id;
    std::string name;
    std::vector<std::string> pets;
};
```

106320 lines of AST

2370 `CXXRecordDecl` *nodes*

1004 `FieldDecl` *nodes*

We need a better plan

```
struct __attribute__((annotate("reflect"))) User
{
    __attribute__((annotate("reflect"))) uint64_t id;
    __attribute__((annotate("reflect"))) string name;
    __attribute__((annotate("reflect"))) vector<string> pets;
};
```

```
#define CLASS() class __attribute__((annotate("reflect-class")))
#define PROPERTY() __attribute__((annotate("reflect-property")))

CLASS() User
{
public:
    PROPERTY()
    uint64_t id;

    PROPERTY()
    string name;

    PROPERTY()
    vector<string> pets;
};
```

```
ClassFinder classFinder;
MatchFinder finder;

DeclarationMatcher classMatcher =
    cxxRecordDecl(decl().bind("id"), hasAttr(attr::Annotate));
DeclarationMatcher propertyMatcher =
    fieldDecl(decl().bind("id"), hasAttr(attr::Annotate));
DeclarationMatcher functionMatcher =
    functionDecl(decl().bind("id"), hasAttr(attr::Annotate));

finder.addMatcher(classMatcher, &classFinder);
finder.addMatcher(propertyMatcher, &classFinder);
finder.addMatcher(functionMatcher, &classFinder);
```

```
struct ClassFinder : public MatchFinder::MatchCallback
{
    virtual void run(MatchFinder::MatchResult const &result);

    virtual void onStartOfTranslationUnit();

    virtual void onEndOfTranslationUnit();
};
```

```
virtual void
ClassFinder::run(MatchFinder::MatchResult const &result) override
{
    CXXRecordDecl const *record = result.Nodes.getNodeAs<clang::CXXRecordDecl>("id");
    if (record)
        return FoundRecord(record);

    FieldDecl const *field = result.Nodes.getNodeAs<clang::FieldDecl>("id");
    if (field)
        return FoundField(field);

    FunctionDecl const *function = result.Nodes.getNodeAs<clang::FunctionDecl>("id");
    if (function)
        return FoundFunction(function);
}
```

```
void
ClassFinder::FoundRecord(CXXRecordDecl const *record)
{
    record->dump();
}

void
ClassFinder::FoundField(FieldDecl const *field)
{
    field->dump();
}

void
ClassFinder::FoundFunction(FunctionDecl const *function)
{
    function->dump();
}
```

```
int main(int argc char **argv)
{
    /* ... */

    return tool.run(newFrontendActionFactory(&finder).get());
}
```

```
CXXRecordDecl 0x7fcda1bae7e0 <./metareflect.hxx:19:24, test.hxx:130:1> line:115:9 class User definition
|-DefinitionData aggregate standard_layout
| |-DefaultConstructor exists non_trivial needs_implicit
| |-CopyConstructor simple non_trivial has_const_param needs_overload_resolution implicit_has_const_param
| |-MoveConstructor exists simple non_trivial needs_overload_resolution
| |-CopyAssignment non_trivial has_const_param needs_implicit implicit_has_const_param
| |-MoveAssignment exists simple non_trivial needs_overload_resolution
| `-Destructor simple non_trivial needs_overload_resolution
|-AnnotateAttr 0x7fcda1bae908 <./metareflect.hxx:19:45, col:83> "reflect-class;"
|-CXXRecordDecl 0x7fcda1bae960 <col:24, test.hxx:115:9> col:9 implicit class User
|-AccessSpecDecl 0x7fcda1bae9f8 <line:118:1, col:7> col:1 public
|-FieldDecl 0x7fcda1baea80 <./metareflect.hxx:21:27, test.hxx:121:14> col:14 id 'uint64_t':'unsigned long long'
| `-AnnotateAttr 0x7fcda1baeac8 <./metareflect.hxx:21:42, col:83> "reflect-property;Serialized"
|-FieldDecl 0x7fcda1baebb0 <col:27, test.hxx:125:12> col:12 name 'string':'std::__1::basic_string<char>'
| `-AnnotateAttr 0x7fcda1baebf8 <./metareflect.hxx:21:42, col:83> "reflect-property;Serialized"
|-FieldDecl 0x7fcda227a228 <col:27, test.hxx:129:20> col:20 pets
    'vector<string>':'std::__1::vector<std::__1::basic_string<char>, std::__1::allocator<std::__1::basic_string<char> > >'
| `-AnnotateAttr 0x7fcda227a270 <./metareflect.hxx:21:42, col:83> "reflect-property;Serialized"
|-CXXConstructorDecl 0x7fcda227a328 <test.hxx:115:9> col:9 implicit User 'void (const User &)' inline default noexcept-unevaluated 0x7fcda227a328
| `-ParmVarDecl 0x7fcda227a460 <col:9> col:9 'const User &'
|-CXXConstructorDecl 0x7fcda227a4f8 <col:9> col:9 implicit User 'void (User &&)' inline default noexcept-unevaluated 0x7fcda227a4f8
| `-ParmVarDecl 0x7fcda227a630 <col:9> col:9 'User &&'
|-CXXMethodDecl 0x7fcda227a6c8 <col:9> col:9 implicit operator= 'User &(User &&)' inline default noexcept-unevaluated 0x7fcda227a6c8
| `-ParmVarDecl 0x7fcda227a7f0 <col:9> col:9 'User &&'
`-CXXDestructorDecl 0x7fcda227a878 <col:9> col:9 implicit ~User 'void ()' inline default noexcept-unevaluated 0x7fcda227a878

FieldDecl 0x7fcda1baea80 <./metareflect.hxx:21:27, test.hxx:121:14> col:14 id 'uint64_t':'unsigned long long'
`-AnnotateAttr 0x7fcda1baeac8 <./metareflect.hxx:21:42, col:83> "reflect-property;Serialized"

FieldDecl 0x7fcda1baebb0 <./metareflect.hxx:21:27, test.hxx:125:12> col:12 name 'string':'std::__1::basic_string<char>'
`-AnnotateAttr 0x7fcda1baebf8 <./metareflect.hxx:21:42, col:83> "reflect-property;Serialized"

FieldDecl 0x7fcda227a228 <./metareflect.hxx:21:27, test.hxx:129:20> col:20 pets
    'vector<string>':'std::__1::vector<std::__1::basic_string<char>, std::__1::allocator<std::__1::basic_string<char> > >'
`-AnnotateAttr 0x7fcda227a270 <./metareflect.hxx:21:42, col:83> "reflect-property;Serialized"
```

```
void
ClassFinder::FoundRecord(CXXRecordDecl const *record)
{
    m_fileName = m_sourceman->getFilename(record->getLocation());
    m_fileName.erase(m_fileName.end() - 4, m_fileName.end());
    m_fileName.append(".generated.hxx");
    m_classes.emplace_back(ReflectedClass(record));
}

void
ClassFinder::FoundField(FieldDecl const *field)
{
    m_classes.back().AddField(field);
}

void
ClassFinder::FoundFunction(FunctionDecl const *function)
{
    m_classes.back().AddFunction(function);
}
```

```
virtual void
ClassFinder::onEndOfTranslationUnit() override
{
    std::error_code ec;
    raw_fd_ostream os(m_fileName, ec);
    assert(!ec && "error opening file");
    for (auto &ref : m_classes)
        ref.Generate(m_context, os);
    m_classes.clear();
}
```

Code Generation

```
void
ReflectedClass::Generate(ASTContext *ctx, raw_ostream &os)
{
    /* ... */
    os << "template<>\n"
       << "Class const *\n"
       << "detail::GetClassImpl(ClassTag<" << type << ">)\n"
       << "{\n"
       << "static Class c;\n";

    FieldGenerator fieldGenerator(ctx, type);
    for (size_t i = 0, n = m_fields.size(); i < n; ++i)
        fieldGenerator.Generate(m_fields[i], i, os);

    os << "}\n";
}
```

```
void
FieldGenerator::Generate(FieldDecl const *field, size_t i, raw_ostream &os)
{
    os << "c.fields[" << i << "].type = "
       << typeName << ";\n";

    os << "c.fields[" << i << "].name = "
       << fieldName << ";\n";

    os << "c.fields[" << i << "].offset = "
       << "offsetof(" << typeName << ", " << fieldName << ");\n";
}
```

Demo

Future

Thank You

Links

→ Working implementation:
github.com/leandros/metareflect

→ Twitter:
twitter.com/ArvidGerstmann

→ My Blog:
arvid.io

Bonus

Class Storage

```
class Class {
    /* ... */
protected:
    Class *m_baseClass;
    Field *m_fields;
    Field *m_fieldsEnd;
    Function *m_functions;
    Function *m_functionsEnd;
    char const *m_name;
    size_t m_flags;
};
```

Class Storage

```
template<class Type, size_t NFields, size_t NFunctions, size_t NTemplateArgs>
struct ClassStorage {
    template<class Lambda>
    ClassStorage(Lambda &&ctor) noexcept
    {
        ctor(this);
    }
    size_t const numFields = NFields;
    size_t const numFunctions = NFunctions;
    size_t const numTemplateArgs = NTemplateArgs;
    Field fields[NFields + 1];
    Function functions[NFunctions + 1];
    TemplateArgument templateArgs[NTemplateArgs + 1];
};
```

LLVM Setup

1. Clone LLVM

```
$ git clone https://git.llvm.org/git/llvm.git/ llvm
```

2. Clone clang into '$LLVM/tools/'

```
$ cd llvm/tools
$ git clone https://git.llvm.org/git/clang.git/
```

3. Clone clang-extra-tools into '$LLVM/tools/clang/tools/extra'

```
$ cd clang/tools
$ git clone https://git.llvm.org/git/clang-tools-extra.git/ extra
```

4. Add your project project

```
$ mkdir yourproject
$ touch yourproject/CMakeLists.txt
$ echo "add_subdirectory(yourproject)" >> extra/CMakeLists.txt
```

5. Generate the project using CMake

```
$ cmake -G"Ninja"
```

Annotations

```
#define CLASS(...) class __attribute__((annotate("reflect-class;" #__VA_ARGS__)))
#define UNION(...) union __attribute__((annotate("reflect-class;" #__VA_ARGS__)))
#define PROPERTY(...) __attribute__((annotate("reflect-property;" #__VA_ARGS__)))
#define FUNCTION(...) __attribute__((annotate("reflect-function;" #__VA_ARGS__)))

CLASS(Serialized) User
{
    PROPERTY(Serialized)
    uint64_t id;

    PROPERTY(Serialized)
    string name;

    PROPERTY(Serialized)
    vector<string> pets;
};
```